NOBUGS2002/017
February 1, 2008

`autotools` for writing portable software

Jens Krüger

ZWE FRM-II
Technische Universität München
Lichtenbergstr. 1
D-85748 Garching, F.R.G.
jens.krueger@frm2.tu-muenchen.de

Most of the scientific software is developed in the same manner: It starts at a lab and the scientist says ”This part of software is only for my private use”. But the software grows and grows, and it comes the time, that other scientists want to use it, too. If they are not working in the same lab and on the same machine, it leads to the problem how to install this software on the new machine. Sometimes it is relatively easy to adopt the `makefile` to the new conditions (compiler, linker, libraries), if it is well written. But mostly it is not done by changing some pathes. One has to change the compiler, linker, additional or standard libraries, and so on. I want to show how you avoid a lot of trouble, if you write software, which may be used on different platforms.

PRESENTED AT

NOBUGS 2002
Gaithersburg, U.S.A, November 04–06, 2002

1 Introduction

If you start to write a software, you cannot imagine that anybody wants to use it, too. It's often that you start to write a part of software for your own use only.

An example: You want to analyse your experimental data. Your data are stored in an electronic form, and you want to find out the maximum count for each spectrum you measured. The first approach is to look at the files if they are stored in a human readable format and find out the maximum by searching by hand. It is a stupid job. This may also been done by a computer. It is foolish enough to do this job if you tell it, what is to be done. The next step is to write all maxima for all files you analysed and draw a plot with these values. The written program helps you to analyse the data. You tell this to one of your colleagues who wants to use it, too. But his machine is not same as yours. He has a different operation system, compiler, etc.

To install this software on his machine you have to change your `Makefile` (which hopefully exists). Afterwards you try to create your software on this machine. After some efforts you may finish this job. But actually you would like to have a system doing this job for you. You remembered that in some software packages you downloaded from the WWW they told you to install a package by typing in only the following three commands:

```
user> ./configure
user> make
user> make install
```

and the software is installed and useable.

You wondered, where this mysterious `configure` comes from and who wrotes this nearly incomprehensible `Makefile`, since your own `Makefile` look much simpler. The wizards who have done these jobs are: the `autotools`.

This set of tools comes from the GNU world. It consists of three parts: `autoconf`, `automake`, and `libtool`. `autoconf` is designed for checking the system (testing compiler, linker, its options and so on), `automake` for creating a makefile with automatically added standard rules from a simple template file, and `libtool` assists the developer during the creation of statically and/or dynamically linked libraries. All three may be used alone, but the combination of all of them is the most efficient way.

I will give an introduction of using the `autotools` in combination. I will show, how you can write a software which is portable from one to another system.

2 Example

Let's consider the following situation. You have some files belonging to the project (some of these files build a library):

```
main.c
func_a.h
func_a.c
func_b.h
func_b.c
```

The file `main.c` contains the code for the main program whereas `func_a.c` and `func_b.c` contain the code for two functions `A()` and `B()` .

The files `func_a.h` and `func_b.h` contain the declarations of functions `A()` and `B()`.

You want to generate a program `myprog` and a library `libmylib`. The library may be created as static as well as as shared library.

You have to write two files, `configure.ac` and `Makefile.am` for doing this job. As you can imagine the file `configure.ac` will contain some information for the configuration of the software and the `Makefile.am` some information for building the software from the sources.

At first let's have a look at the `configure.ac` file.

3 `configure.ac` file

The `configure.ac` file starts with a line:

```
AC_INIT(myproject, 1.0.0, [myproject-bug@myproject.org])
```

This lines tells the system: Generate a project named `myproject` with the version number `1.0.0`. The last parameter gives the email address by which the developer of this project may be contacted.

The meaning of the next three lines is the following:

```
AC_PREREQ(2.52)
AC_COPYRIGHT([(c) Jens Krueger])
AC_REVISION([1.0])
```

the system requires an autoconf version at least 2.52, it defines the copyright message and defines the version of this file.

By the help of

```
AC_CONFIG_SRCDIR(func_a.c)
```

the system may verify that it works on the right system. The parameter of `AC_CONFIG_SRCDIR` should point to a file unique for this project.

```
AM_INIT_AUTOMAKE([1.6])
```

gives `autotools` the input that at least the version 1.6 of `automake` is required.

So up to this point you have configured a lot of things only for the `autotools` itself, but nothing for your own project with the exception of the first line. For compiling your project you need a C compiler. To tell `autotools`: "Search for a C compiler and linker and find out the right options" you use the following line.

```
AC_PROG_CC
```

The next line checks the existence of `libtool` which is very helpful for creating libraries.

```
AC_PROG_LIBTOOL
```

An useful feature of `autotools` is the preselection of an installation path. As developer you may preselect a path for the installation of the software package. If the user does not overwrite this during the configuration run, the default installation path will be used. The line

```
AC_DEFAULT_PREFIX(/usr/local)
```

sets the default installation path. This may be overwritten during the configuration time with the option `--prefix=/new-installation-path`.

The last lines are very interesting for the project. These lines tell the `autotools` which files have to be created. The `AC_CONFIG_FILES` macro defines the file names to be created. For each file given in this list, a file with the extension "`.in`" has to exist. In your example you need a file `Makefile.in`. But where does it come from? It comes from the `Makefile.am`. How this works will be explained in the next chapter. `AC_OUTPUT` macro performs the creation of the files given by the `AC_CONFIG_FILES` macro.

```
AC_CONFIG_FILES(Makefile)
AC_OUTPUT
```

4 `Makefile.am` file

As described in the previous chapter, the `Makefile.am` leads to a `Makefile` which can be used by the `make` command.

The `Makefile.am` contains the following lines:

```
bin_PROGRAMS = myprog
myprog_SOURCES = main.c
lib_LTLIBRARIES = libmylib.la
libmylib_la_SOURCES = func_a.c func_b.c
include_HEADERS = func_a.h func_b.h
```

Now let's have a deeper look into the Makefile.am. The first line declares that you want to generate an executeable, which should be installed in the $prefix/bin directory, called myprog. The following line gives information on the source file(s) from which the myprog should be generated: main.c. If the myprog should also be generated from file myprog.c you have simply to add myprog.c to this line.

The third line tells you that you want to build a so-called libtool archive (.la). This is a library, but at this time you have not to decide whether you want to use static or shared libraries. This will be done at configuration time. The library will be installed in the $prefix/lib directory.

The fourth line defines the sources from which the library has to be built up. Please pay attention to the fact, that the '.' has been replaced by a underscore character.

The last line defines that the files func_a.h and func_b.h have to be installed in the $prefix/include directory.

5 Preparing the system for a build

Now you have finished the set-up of your project and you should start the creation of the configure file.

By calling

```
user> libtoolize --copy --automake
```

you tell the libtool package that you want to use it in cooperation with the automake package and it should copy all neccessary files into the local directory. Omitting the parameter --copy the libtoolize command only creates symbolic links instead of copies of its administrative files.

The next step you have to do is to call the aclocal program, which extracts all macros used in the configure.ac and generates a file aclocal.m4 later used by autoconf.

The third call

```
user> automake --copy --add-missing
```

generates a template Makefile.in and installs some administrative files. Omitting the parameter --copy similarly to the libtoolize call leads to symbolic links instead of local copies of the admin files. You get some messages:

```
Makefile.am: required file ‘./NEWS’ not found
Makefile.am: required file ‘./README’ not found
Makefile.am: required file ‘./AUTHORS’ not found
Makefile.am: required file ‘./ChangeLog’ not found
```

These missing files are required in the GNU world. It is a good idea to create these files by

```
user> touch NEWS README AUTHORS ChangeLog
```

and fill them with useful information about your project. Additionally `automake` creates two other files `COPYING` and `INSTALL` which contain the GNU Public License and the standard installation guide of the project. If you want to change the license or the installation guide simply change the content of these files. If you created them as symbolic links remove them and create regular files with the same name.

At last you have to call

```
user> autoconf
```

which produces the `configure` script.

The preparations for generating a portable build process are finished. Now you configure your system by typing in:

```
user> ./configure
```

The output of this command tells you a lot of things, for example what kind of C compiler is installed and found on the system, whether the system may create static or shared libraries, whether they should be created, and so on. If the call does finish without any errors you may start your build process by typing in

```
user> make
```

If the build process ran successfully, you may install your package by typing:

```
user> make install
```

The reverse process may be called by typing

```
user> make uninstall
```

6 Roll out

If your package runs correctly you need a way for distributing your software in a tar'ed format. The simplest but not the best way is to tar the complete directory, in which you built your software. The `autotools` help you with two targets in the `Makefile` which are generated automatically: `dist` and `distcheck`.

The `dist` target provides the creation of a `tar` file containing all necessary files for a build process on another machine. The created `tar` file has to be installed on the target machine. But this way is not safe. A better way is to call the target `distcheck`.

This target creates a `tar` file, installs this in a temporary directory, calls the `configure` script and the `make` command to build the software, to install, and to uninstall it. If the process was successful you get the message:

```
================================================
myproject-1.0.0.tar.gz is ready for distribution
================================================
```

This message tells you that your software package distributed in the `myproject-1.0.0.tar.gz` file may be installable on other machines with different operating systems and architectures.

7 Extending the configuration task

At this point we have to make some remarks: Above a very simple example was discusssed to show you how to use the `autotools`. They are powerful and provide a lot of macros for distinct tasks. So it is possible:

- to search for a special library
- to search for a special function in a library
- to look for a special header file on the target system
- and, and, ...

`autotools` also give you the chance to create a machine dependent config file which may be included in your sources. This controls the compile process of the sources depending on the found header files, compilers, library versions and so on.

Let's extend your project by some of these features. Assuming you want to use the NeXus library [9] in your project some requirements have to be fulfilled. You have to install the HDF library [8] on which NeXus is based and the NeXus library itself.

What do you have to do now? For the inclusion of NeXus you need the declaration of the NeXus API which is given in the file `napi.h` as well as the library `libNeXus` either in statically or dynamically linked form. For the HDF library you need the HDF API, which is given in the file `mfhdf.h` and the library itself `libmfhdf`. With this knowledge you may tell your configuration system that it has to look for these files.

This has to be done in the `configure.ac` file. Please insert after line 9 the following lines:

```
AC_CHECK_HEADERS([napi.h mfhdf.h],
        [],
        AC_MSG_ERROR([Did not found the NeXus headers]))
```

These lines instruct the `autotools` to search in the standard include pathes for the files `napi.h` and `mfhdf.h`. If one of these file was not found the `configure` script stops with an error message:

```
configure: error: Did not found the NeXus headers.
```

For searching the HDF library you have to add these lines into your file `configure.ac`:

```
AC_CHECK_LIB(mfhdf, Hclose,
        AC_MSG_RESULT([HDF library found]),
        AC_MSG_ERROR([Did not found the HDF library]),
[-ldf -ljpeg -lz])
```

It is designed to search for a special function (in this case `Hclose`) in the library `libmfhdf` to test the existence of the library. If it is found `configure` outputs:

```
HDF library found.
```

or else it gives an error message that it did not found the library and stops its execution.

For the search of the NeXus library you should add:

```
AC_CHECK_LIB(NeXus, nxiclose_,
         [],
         AC_MSG_ERROR([Did not found the NeXus library]))
```

The `AC_CHECK_LIB` assumes that the tested function has no parameters. It tests only the linking of the program, if you want to test the right parameters of a certain function you have to perform more sophisticated tests. For this a deeper look into the manuals [2], [4],[6] or the book [1] helps.

8 Summary

This paper gives a short introduction into the use of `autotools`, i.e. `autoconf`, `automake`, and `libtool`. At hand of a little example its usage is explained, and it is discussed how the project may be extended and adopted to the specific environment. The use of autotools simplifies the creation and distribution of software which has to run on different systems.

References

[1] G.V. Vaughan, B. Elliston, T. Tromey, and I.L. Taylor, GNU AUTOCONF, AUTOMAKE, AND LIBTOOL, New Riders, 2001, http://www.newriders.com/

[2] D. MacKenzie, B. Elliston, and A. Demaille, GNU autoconf manual

[3] http://www.gnu.org/software/autoconf/

[4] D. MacKenzie, and T. Tromey, GNU Automake manual

[5] http://www.gnu.org/software/automake/

[6] G. Matzigkeit, A. Oliva, Th. Tanner, and G.V. Vaughan. GNU libtool manual

[7] http://www.gnu.org/software/libtool/

[8] http://hdf.ncsa.uiuc.edu/

[9] http://www.neutron.anl.gov/nexus/